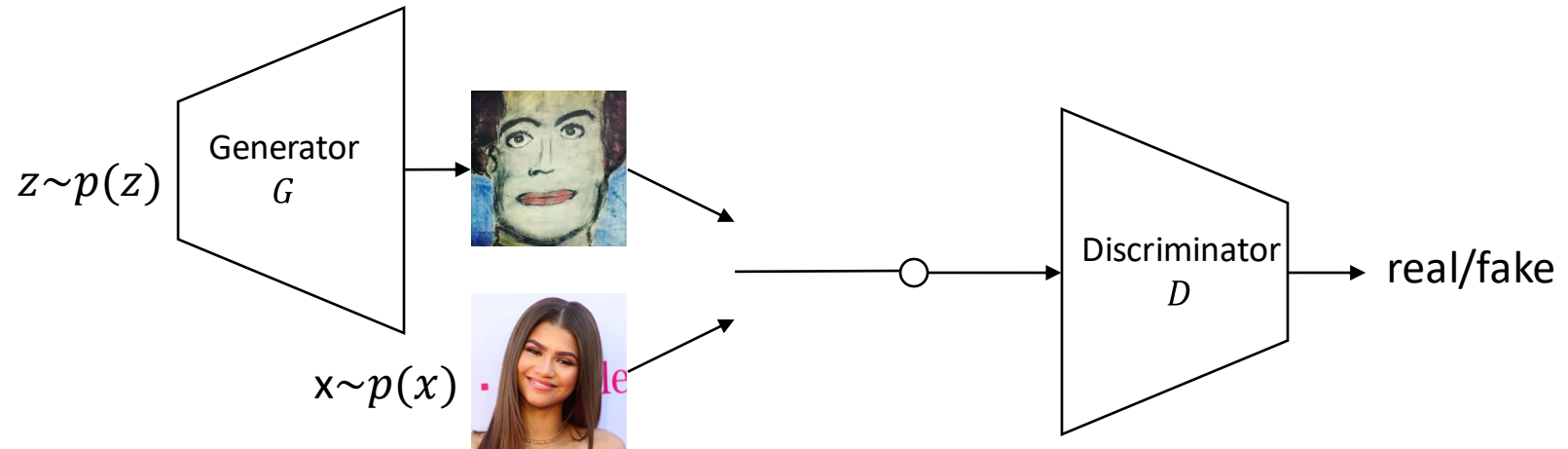


# Normalizing Flows/ Invertible Models

Lecture 7

18-789

# Recap: Generative Adversarial Networks



In practice, we update G/D alternately!

$$\min_G \max_D E_{x \sim p(x)} [\log D(x)] + E_{z \sim p(z)} [\log(1 - D(G(z)))]$$

Generated sample

Discriminator tries to classify real vs fake  
Generator tries to fool the discriminator

# Recap: Theory of GAN

- For a fixed generator  $G$  (with parameter  $\theta$ ), the optimal discriminator is

$$D^*(x) = \frac{p(x)}{p(x) + p_\theta(x)}$$

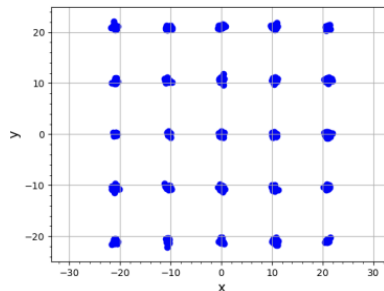
- Assuming optimal discriminator, the generator is trained to minimize JSD (Jenson-Shannon Divergence)

$$\mathcal{L}(G) = 2\text{JSD}(p(x)|p_\theta(x)) - \log 4$$

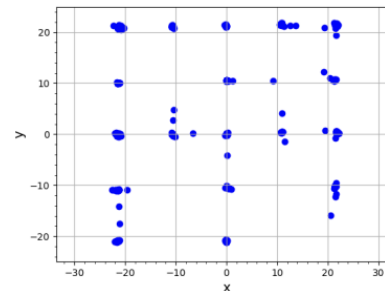
$$\min_G \max_D E_{x \sim p(x)} [\log D(x)] + E_{z \sim p(z)} [\log(1 - D(G(z)))]$$

# Recap: Wasserstein GAN

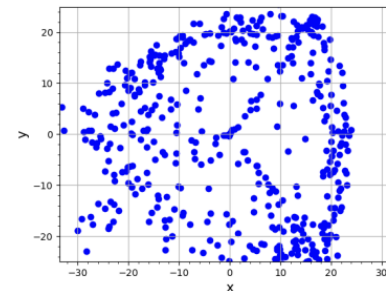
- Motivation: JSD is constant when two distributions have non-overlapped support
- Solution: Wasserstein distance
  - $$W(p, p_\theta) = \inf_{\gamma \in \Pi(p, p_\theta)} E_{(x, y) \sim \gamma} [ ||x - y|| ]$$
$$= \sup_{||D||_L \leq 1} E_{x \sim p} [D(x)] - E_{x \sim p_\theta} [D(x)]$$
- Empirical results not very good
- Lesson: GANs do not really minimize a specific divergence



(a) True data



(b) GAN



(d) WGAN

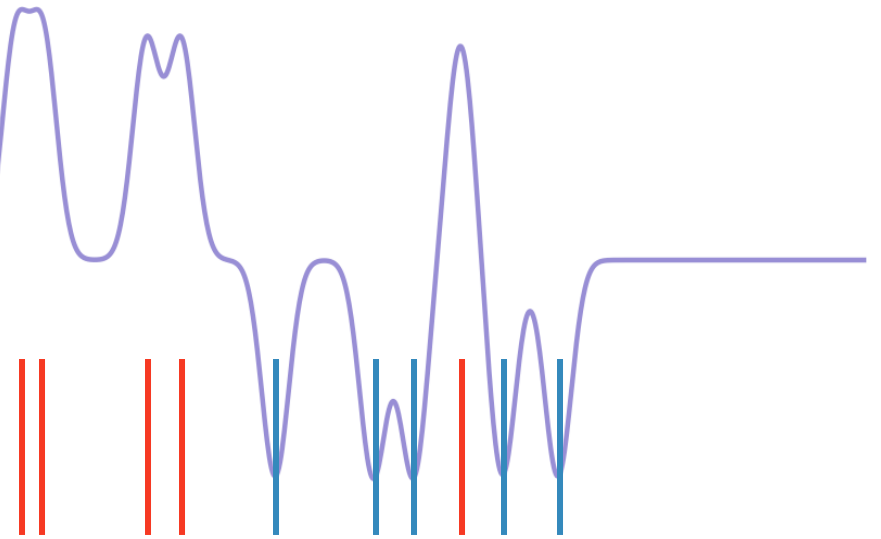
# Recap: Improving Stability of Training GANs

- Key: prevent discriminator overfitting
  - Data augmentation
  - Lipschitz continuity
    - Gradient Penalty (can be applied on any GAN loss!)
    - Spectral Normalization

$$\frac{|f(x) - f(y)|}{|x - y|} \leq K, \forall x, y$$

Large Lipschitz constant  
around “spikes”!

— Real Samples  
— Fake Samples  
— Discriminator Output



# Recap: Improving Stability of Training GANs

- Key: prevent discriminator overfitting
  - Data augmentation
  - Lipschitz continuity
    - Gradient Penalty (can be applied on any GAN loss!)
    - Spectral Normalization
- Other tricks:
  - Adam  $\beta_1 = 0$  for G&D (No momentum)

$$m_w^{(t+1)} := \beta_1 m_w^{(t)} + (1 - \beta_1) \nabla_w L^{(t)}$$

$$w^{(t+1)} := w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \varepsilon}$$

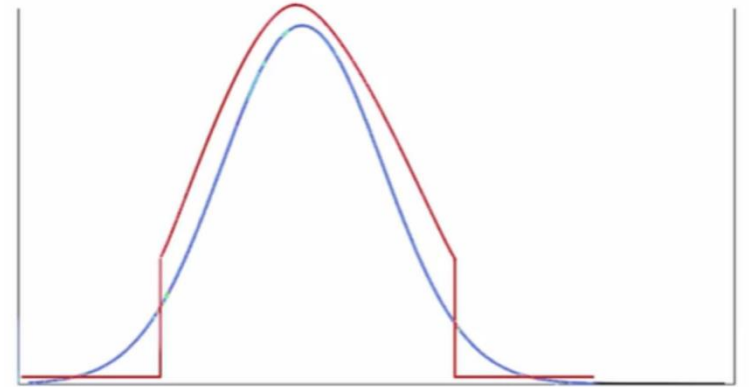
# Recap: Improving Stability of Training GANs

- Key: prevent discriminator overfitting
  - Data augmentation
  - Lipschitz continuity
    - Gradient Penalty (can be applied on any GAN loss!)
    - Spectral Normalization
- Other tricks:
  - Adam  $\beta_1 = 0$  for G&D (No momentum)
  - Exponential Moving Average for G

$$\theta_{EMA}^{(t)} = \beta \theta_{EMA}^{(t-1)} + (1 - \beta) \theta^{(t)}$$

# Recap: Improving Stability of Training GANs

- Key: prevent discriminator overfitting
  - Data augmentation
  - Lipschitz continuity
    - Gradient Penalty (can be applied on any GAN loss!)
    - Spectral Normalization
- Other tricks:
  - Adam  $\beta_1 = 0$  for G&D (No momentum)
  - Exponential Moving Average for G
  - Truncation trick for G

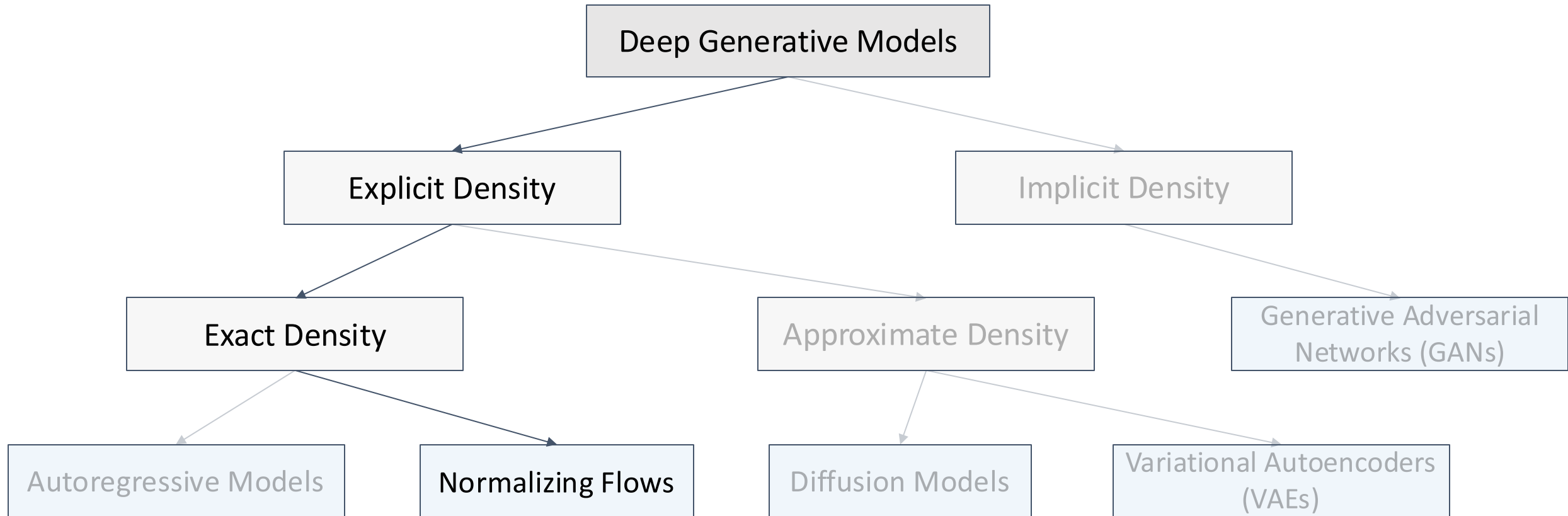




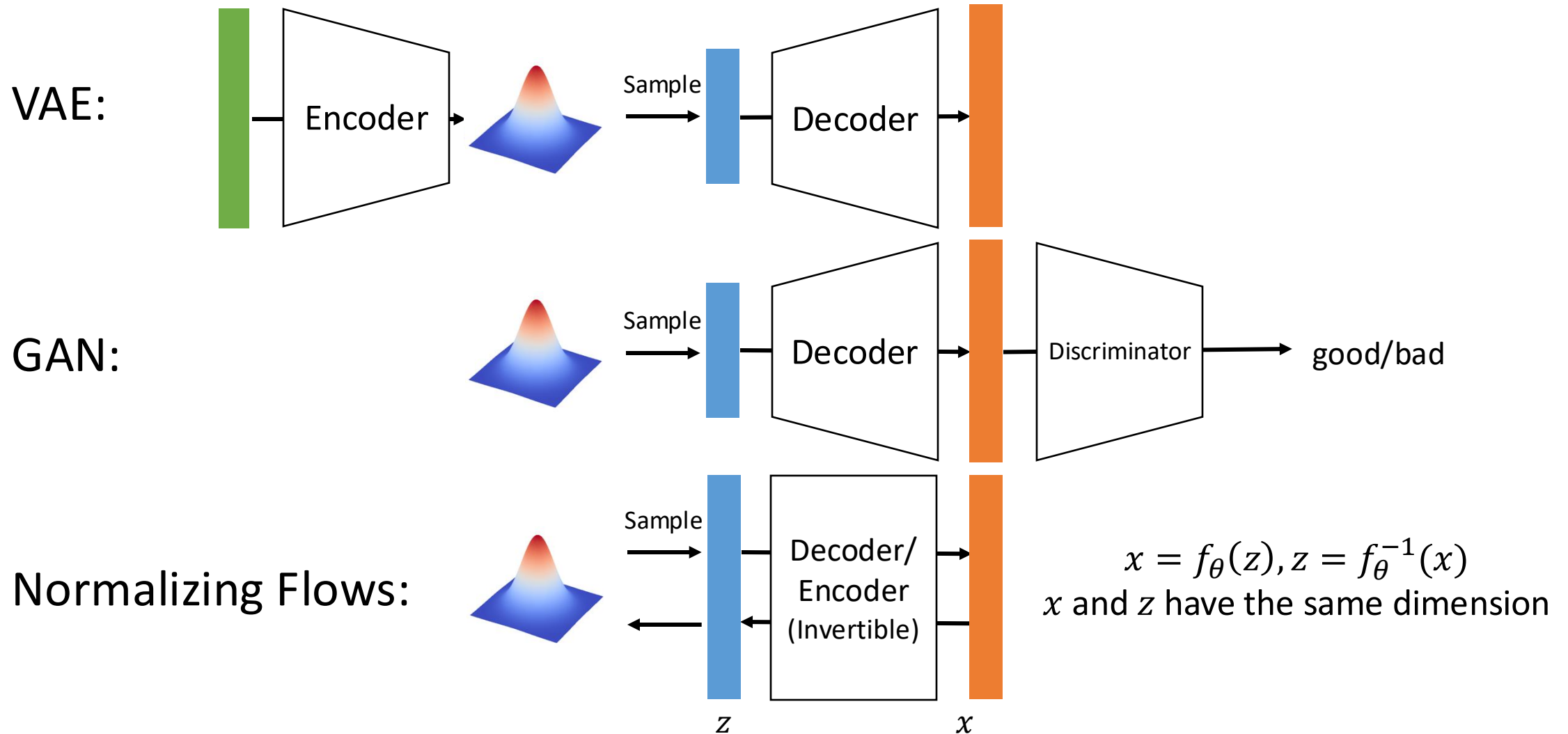
# Deep Generative Models (so far)

- Autoregressive Models: **Generation not parallelizable**
- VAE: **Blurry**
- GAN: **Unstable training**
- Normalizing Flows: **Stable training, parallelized and sharp generation**

# Normalizing Flows



# Normalizing Flows

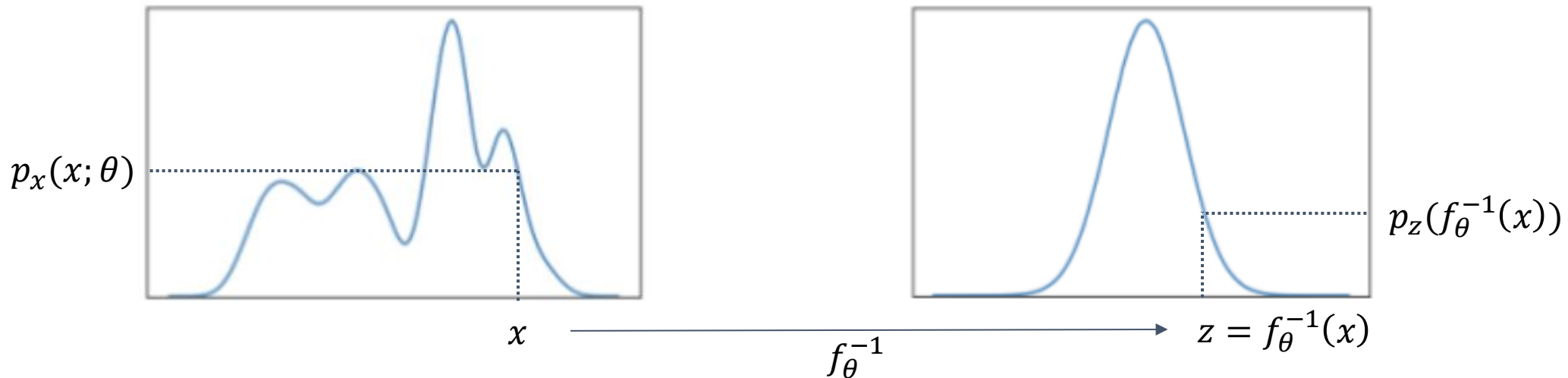


# Change of Variables Theorem

- $x = f_{\theta}(z), z = f_{\theta}^{-1}(x)$

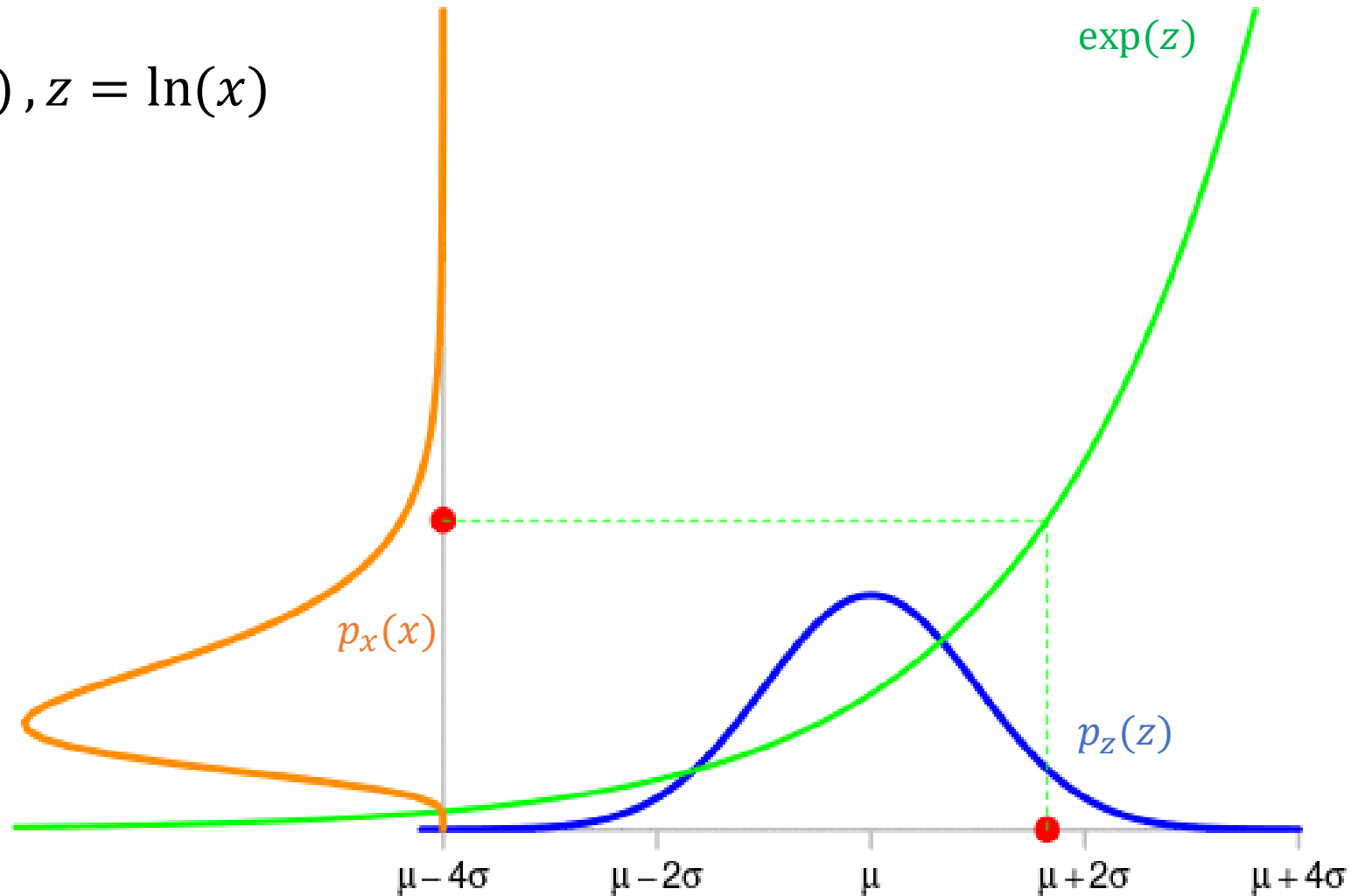
- $p_x(x; \theta) = p_z(f_{\theta}^{-1}(x)) \left| \det \frac{\partial f_{\theta}^{-1}(x)}{\partial x} \right|$

Density of the encoded  $z$  under the prior      Jacobian determinant of  $f_{\theta}^{-1}$  (For 1D: simply the derivative of  $f_{\theta}^{-1}$ )



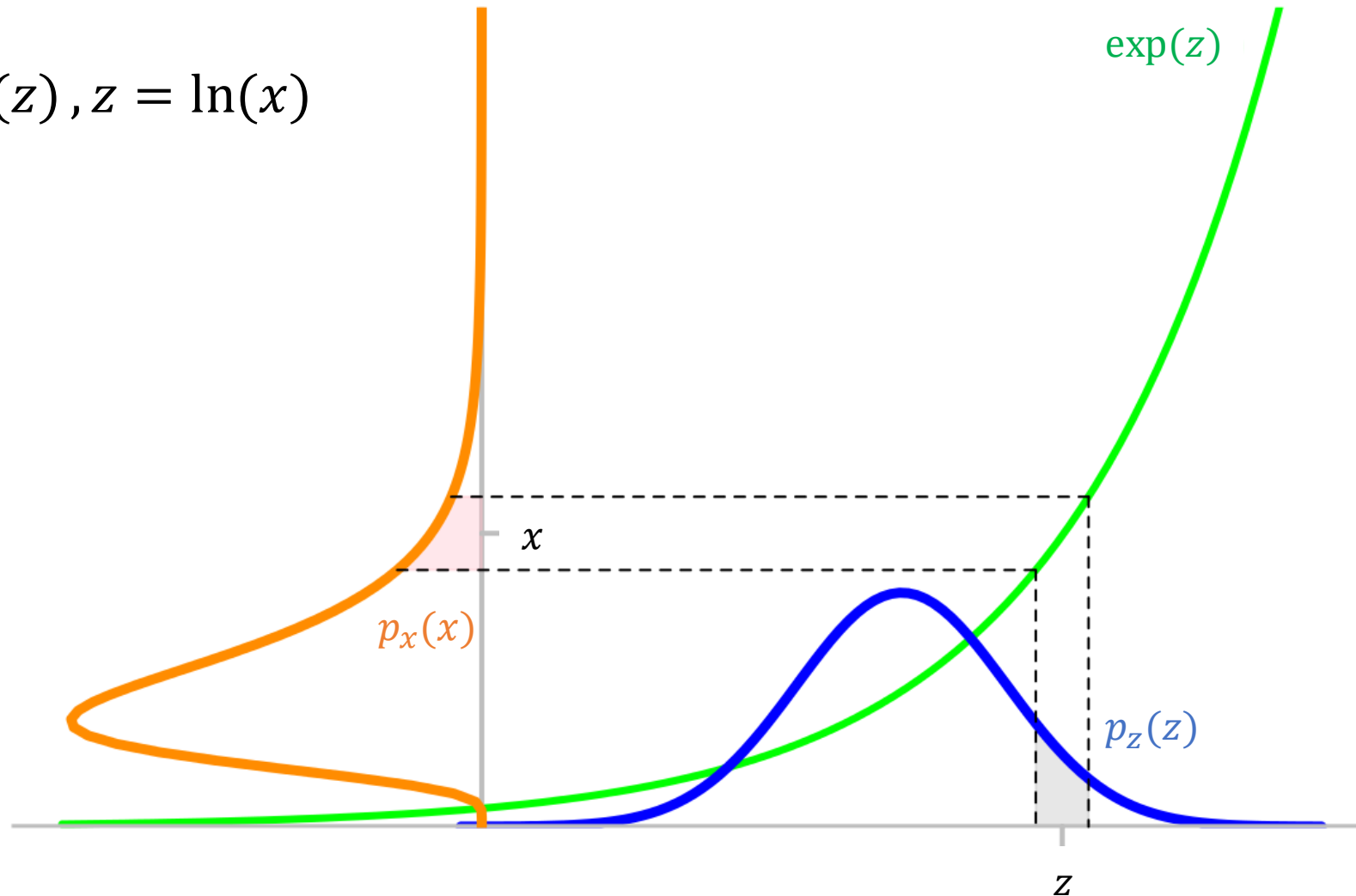
# Change of Variables: Intuition

- $x = \exp(z)$ ,  $z = \ln(x)$



# Change of Variables: Intuition

- $x = \exp(z)$ ,  $z = \ln(x)$

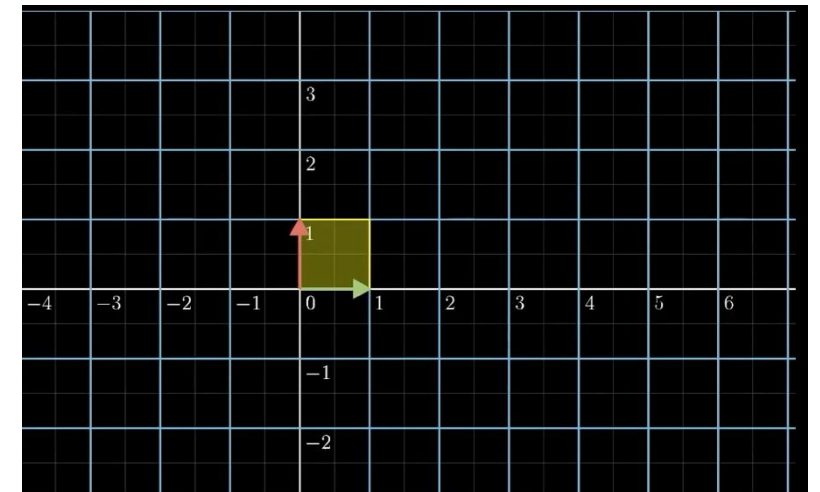
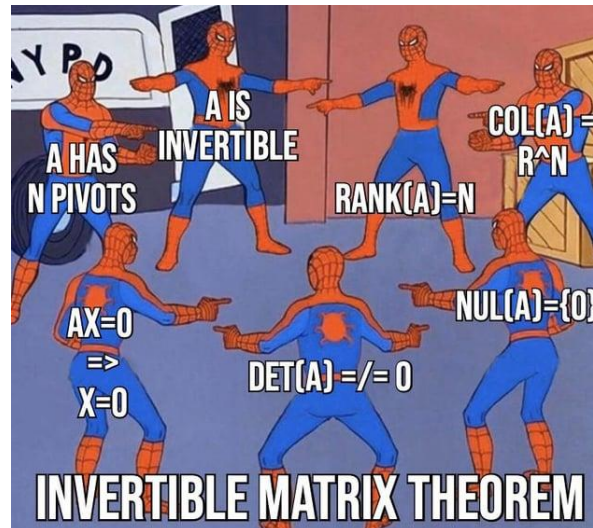


# Jacobian Determinant

$$p_x(x; \theta) = p_z(f_\theta^{-1}(x)) \left| \det \frac{\partial f_\theta^{-1}(x)}{\partial x} \right|$$

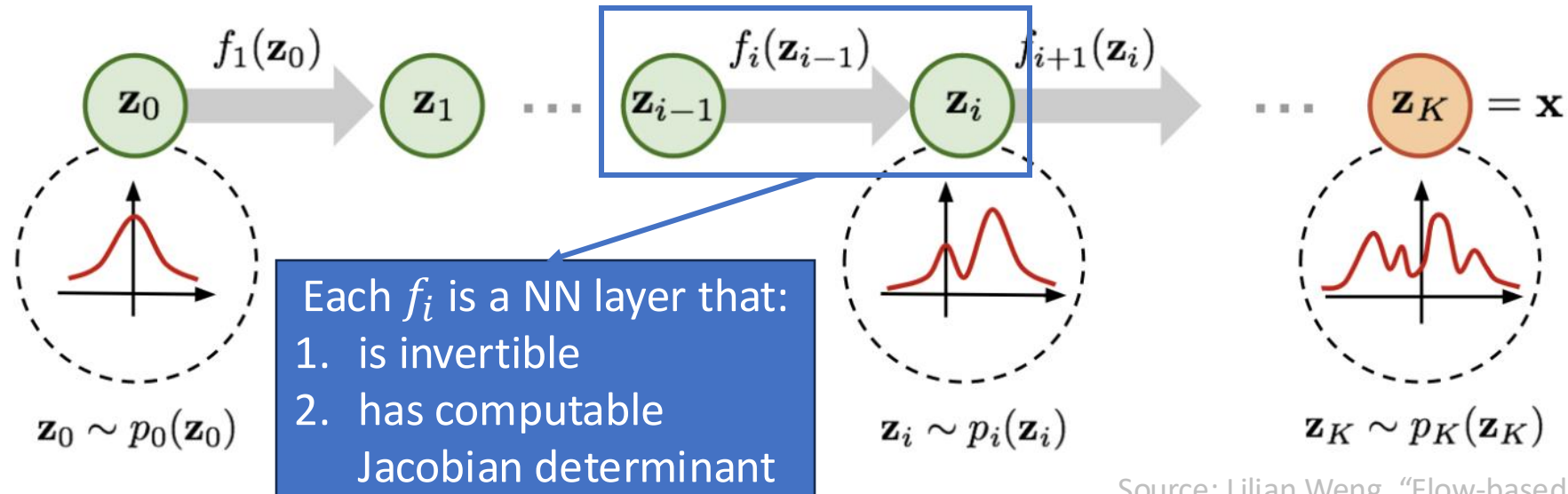
- Jacobian matrix: Local linear approximation of  $f$  (around  $x$ )
- Jacobian *Determinant*: How much does  $f$  “stretches” ( $> 1$ ) or “compresses” ( $< 1$ ) space (around  $x$ )
  - Example:  $J = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$
  - What’s the determinant of  $J$ ?
- $\det J \neq 0 \iff J$  is invertible

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$



# Normalizing Flows

- Normalizing Flows: a **sequence** of  $K$  invertible transformation
  - $f = f_K \circ f_{K-1} \circ \dots \circ f_2 \circ f_1$
- $p_x(x; \theta) = p_z \left( f_\theta^{-1}(x) \right) \prod_{i=1}^K \left| \det \frac{\partial f_i^{-1}}{\partial z_i} \right| = p_z \left( f_\theta^{-1}(x) \right) \prod_{i=1}^K \left| \det \frac{\partial f_i}{\partial z_{i-1}} \right|^{-1}$
- $\log p_x(x; \theta) =$





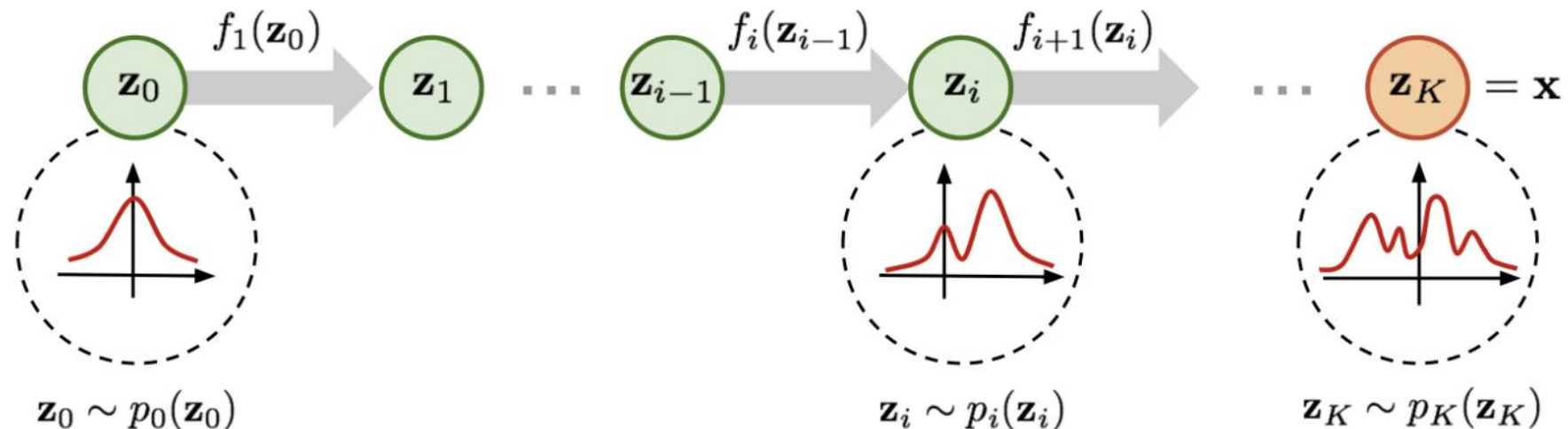
# Normalizing Flows

1. How to model the **joint** distribution of **high-dimensional** data?

- $p_{\theta}(x): x = f_{\theta_K} \circ f_{\theta_{K-1}} \circ \dots \circ f_{\theta_2} \circ f_{\theta_1}(z)$ , where  $z$  is the same dimension as  $x$
- $p(z) = N(0, I)$  is standard Gaussian

2. How to **optimize** your model?

- Maximize Likelihood

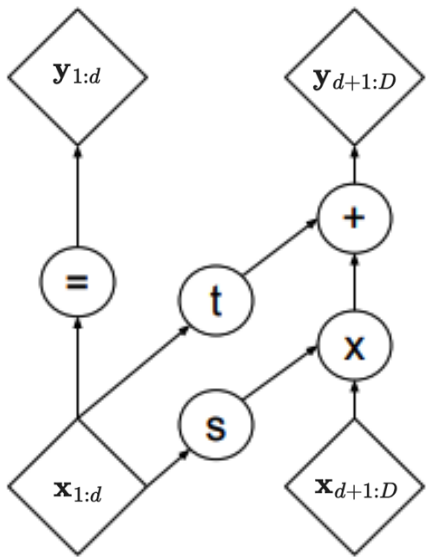


# Example: affine coupling layer

Each  $f_i$  is a NN layer that:

1. is invertible
2. has computable Jacobian determinant

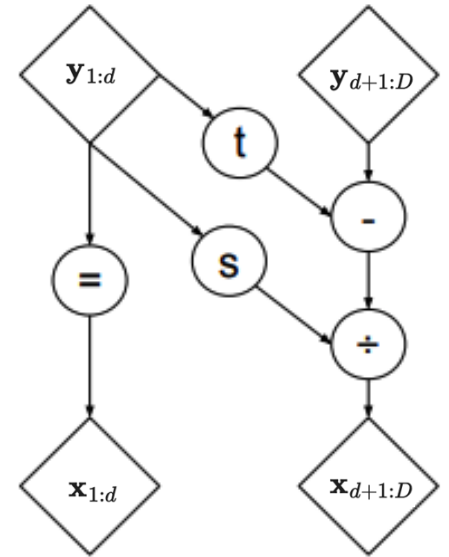
- Split the input along certain dimension (e.g. channel)
  - The first part stays the same
  - The second part undergoes an affine transformation and both scale and shift parameters are predicted by the first part with an FC layer



(a) Forward propagation

$$\begin{aligned}
 \mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\
 \mathbf{y}_{d+1:D} &= \mathbf{x}_{d+1:D} \odot \exp(s(\mathbf{x}_{1:d})) + t(\mathbf{x}_{1:d}) \\
 \mathbf{x}_{1:d} &= \mathbf{y}_{1:d} \\
 \mathbf{x}_{d+1:D} &= (\mathbf{y}_{d+1:D} - t(\mathbf{y}_{1:d})) \odot \exp(-s(\mathbf{y}_{1:d}))
 \end{aligned}$$

$$\mathbf{J} = \begin{bmatrix} \mathbb{I}_d & \mathbf{0}_{d \times (D-d)} \\ \frac{\partial \mathbf{y}_{d+1:D}}{\partial \mathbf{x}_{1:d}} & \text{diag}(\exp(s(\mathbf{x}_{1:d}))) \end{bmatrix}$$



(b) Inverse propagation

# Think-pair-share:

- Given 2D input and an affine coupling layer defined as follows:

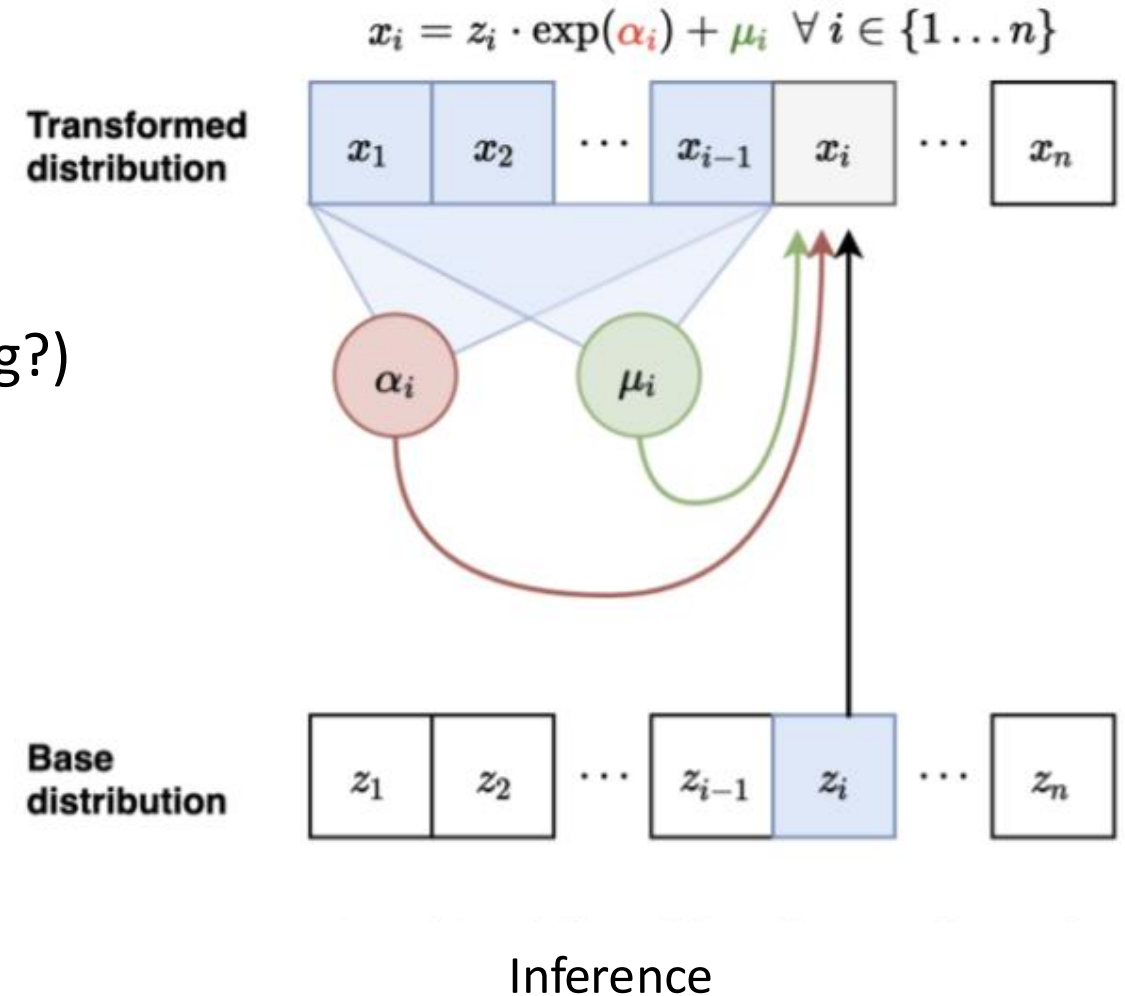
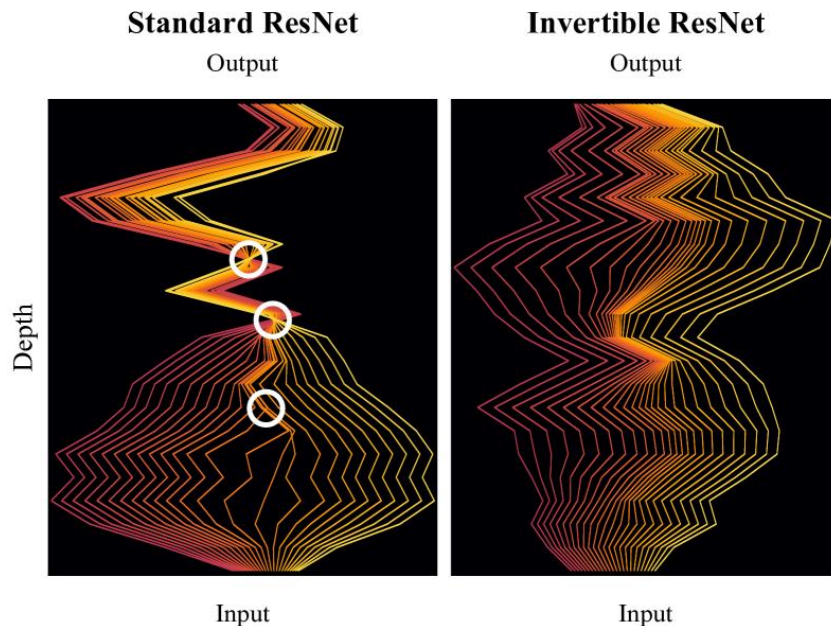
$$\begin{cases} y_1 = x_1 \\ y_2 = x_2 \exp(s(x_1)) + t(x_1) \end{cases}$$

where  $s(x_1) = 0.5x_1 - 1$ ,  $t(x_1) = 2x_1 + 1$

- Q1: Given input  $(x_1, x_2) = (2, 3)$ 
  - compute output  $(y_1, y_2)$
  - What's the Jacobian determinant?
- Q2: Given output  $(y_1, y_2) = (0, 1)$ 
  - compute input  $(x_1, x_2)$

# Other invertible layers

- Permutation [1]
- Autoregressive Flow [2] (vs. affine coupling?)
- Invertible convolution [3]
- Invertible Residual Networks [4]



- [1] "Density estimation using Real NVP", Dinh et al., 2016  
[2] "Masked Autoregressive Flow for Density Estimation", Papamakarios et al., 2017  
[3] "Glow: Generative Flow with Invertible 1x1 Convolutions", Kingma et al., 2018  
[4] "Invertible Residual Networks", Behrmann et al., 2019

# How do Normalizing Flows perform?



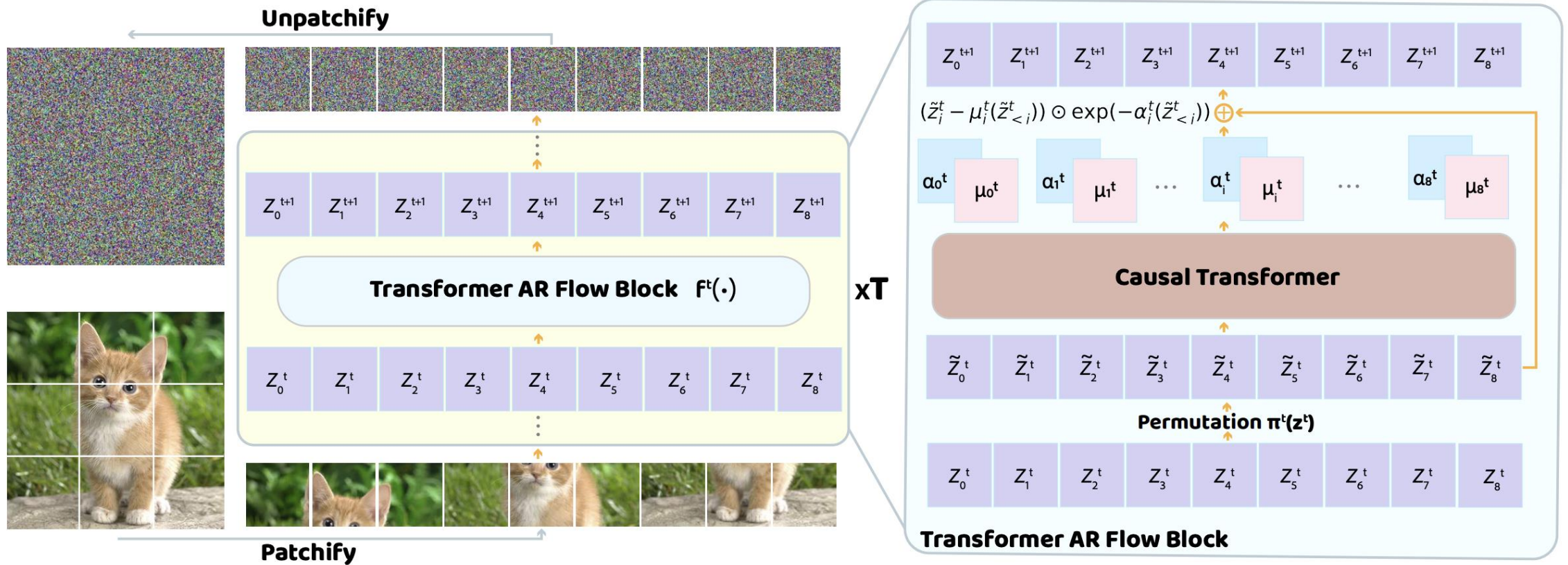
Glow, 2018 (Normalizing Flows)



StyleGAN, 2019 (GANs)



# Modern Normalizing Flows



# Modern Normalizing Flows





# Continuous Normalizing Flows (CNFs)

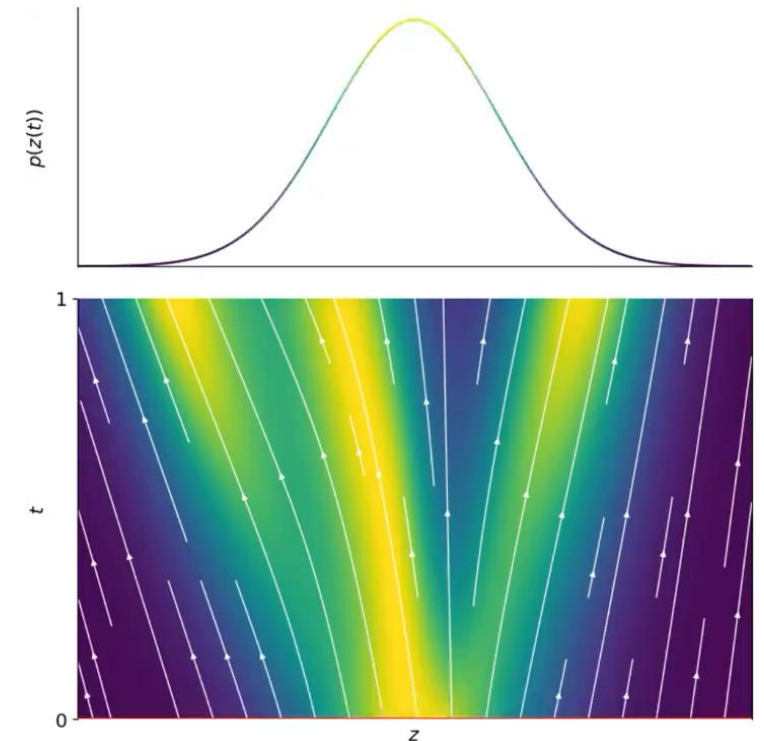
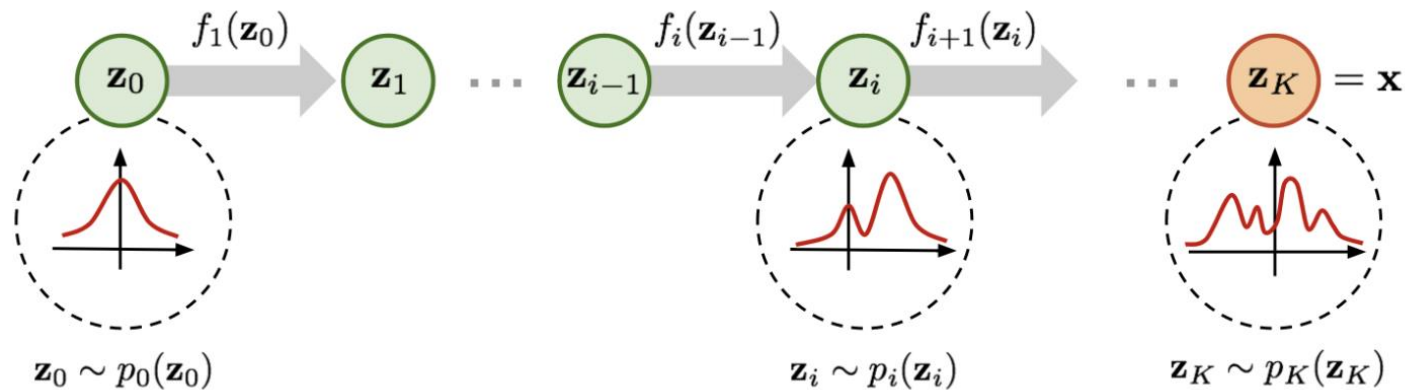
- Normalizing Flows consist of  $K$  discrete transformations

- $z_i = f_i(z_{i-1}), z_K = f_K \circ f_{K-1} \circ \dots \circ f_2 \circ f_1(z_0)$

- Generalize to continuous case

- $\frac{\partial z}{\partial t} = f(z_t, t), 0 < t < 1$

- $z_{t+\nabla t} \approx f(z_t, t)\nabla t + z_t, z_1 = z_0 + \int_0^1 f(z_t, t) dt$





# Continuous Normalizing Flows (CNFs)

- Normalizing Flows consist of  $K$  discrete transformations

- $z_i = f_i(z_{i-1}), z_K = f_K \circ f_{K-1} \circ \dots \circ f_2 \circ f_1(z_0)$

- Generalize to continuous case

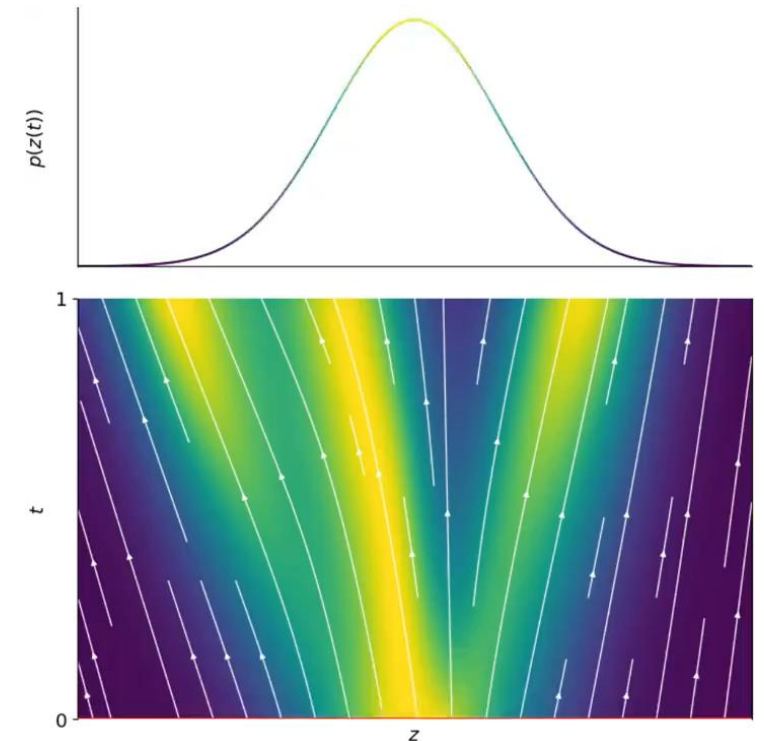
- $\frac{\partial z}{\partial t} = f(z_t, t), 0 < t < 1$

- $z_{t+\nabla t} \approx f(z_t, t)\nabla t + z_t, z_1 = z_0 + \int_0^1 f(z_t, t) dt$

- Training objective

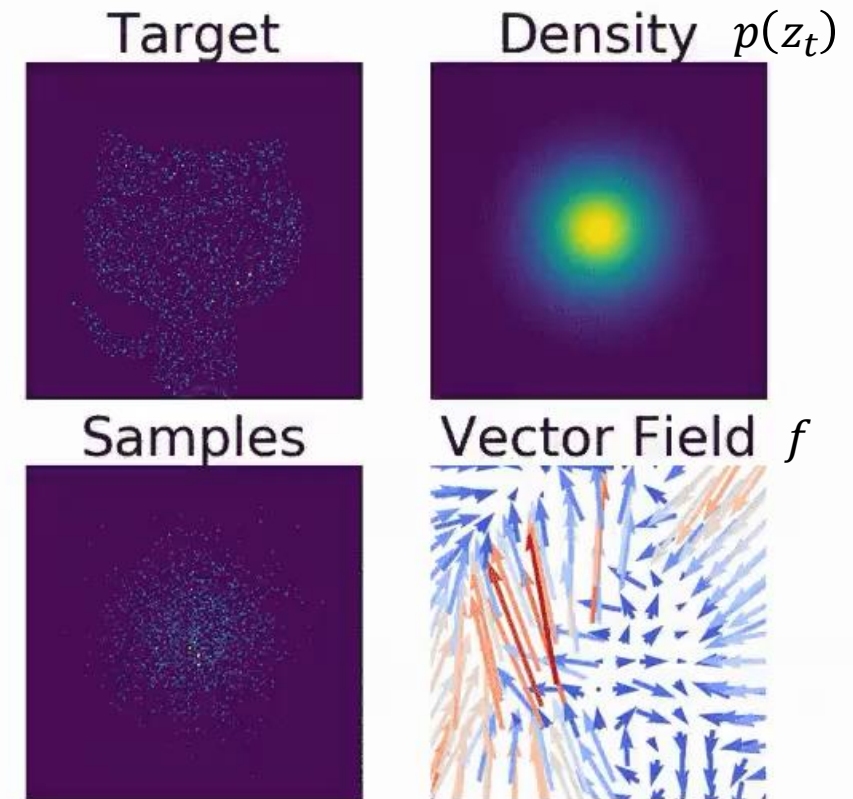
- $\log p(z_1) = \log p(z_0) - \int_0^1 \text{Trace} \left( \frac{\partial f}{\partial z_t} \right) dt$

- Assume  $z_0$  is noise and  $z_1$  is data



# Continuous Normalizing Flows (CNFs)

- Network
  - A neural network  $f(z_t, t)$  conditioned on data  $z_t$  and time  $t$
  - Unrestricted architecture
- Training
  - $\log p(z_1) = \log p(z_0) - \int_0^1 \text{Trace} \left( \frac{\partial f}{\partial z_t} \right) dt$
  - Solve the forward **ODE** to compute log-likelihood
    - Find the latent  $z_0 = z_1 + \int_1^0 f(z_t, t) dt$
    - Estimate the trace  $\int_0^1 \text{Trace} \left( \frac{\partial f}{\partial z_t} \right) dt$
  - Backpropagate through the **ODE**
- Sampling
  - Solve the backward **ODE**
  - $z_1 = z_0 + \int_0^1 f(z_t, t) dt$

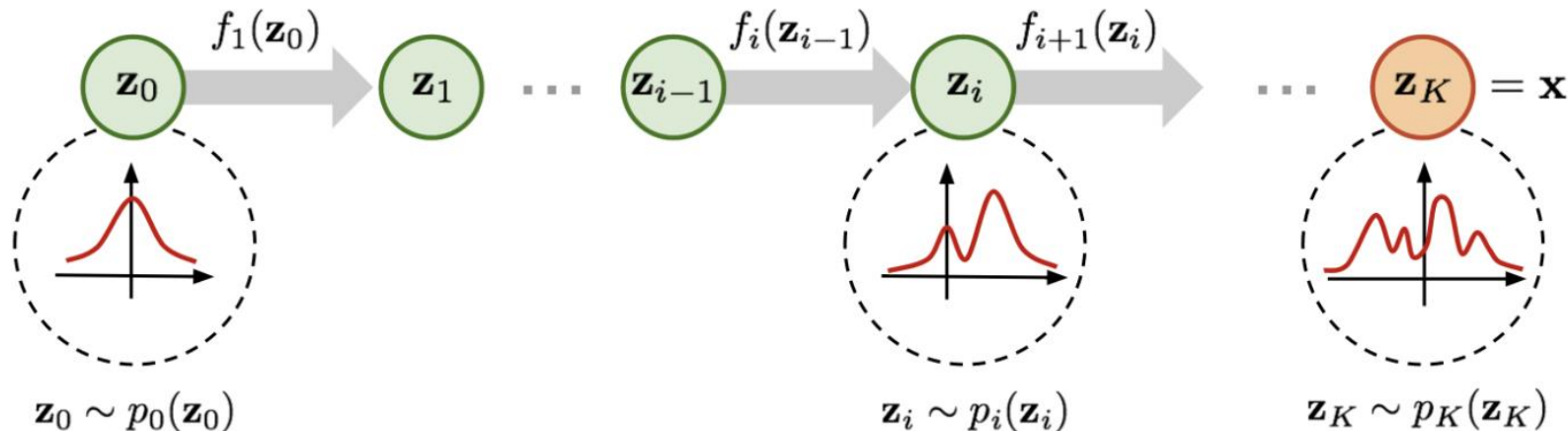


# Pros and Cons

- (Discrete) Normalizing Flows
  - **Different** parameters at different steps
  - **Restricted** (invertible) architecture
- Continuous Normalizing Flows
  - **Same** parameters at different steps
  - **Unrestricted** architecture

***Diffusion*** is a CNF at inference time!  
(but trained in a more efficient way)

Small  $f$  (e.g., single layer) -> **not expressive**  
Large  $f$  (e.g., a large network) -> **slow training**



# 5 Minute Quiz

- On Canvas
- Passcode: snail

